Deep Learning in Game Solving



15 888 **Computational Game Solving** (Fall 2026)

Ioannis Anagnostides

Today’s lecture

- Stochastic regret minimization and **Monte Carlo CFR**
  - General theoretical guarantees
  - External and outcome sampling
  - Sublinear algorithms and sparsity
- **Deep CFR**
  - Function approximation as a general-purpose abstraction technique
  - Approximate version of CFR using deep neural networks
- **ESCHER**
  - Improved gradient estimator by reducing the variance

Limitations of CFR and stochastic regret minimization

- In very large games, even performing a few iterations of CFR can be entirely prohibitive as it requires a *full game-tree traversal*
- Can we reduce the *per-iteration complexity* of CFR without incurring a large degradation in the regret?
- **Stochastic regret minimization** provides an answer to this question
- Monte Carlo CFR (Lanctot et al., 2009) is a special case of this framework in extensive-form games
- *Libratus* uses MCCFR to construct the blueprint strategy

Reminder of CFR

Algorithm 1: Counterfactual regret minimization (CFR)

Input: A regret minimizer $\mathfrak{R}_j$ for each decision point $j \in \mathcal{J}$ of the TFDP.
NextStrategy():
  for *each decision point* $j \in \mathcal{J}$ **do**
    $\Delta(\mathcal{A}_j) \ni b_j^{(t)} \coloneqq \mathfrak{R}_j$.NextStrategy();
  for *each decision point* $j \in \mathcal{J}$ *in top-down order* **do**
    for *each action* $a \in \mathcal{A}_j$ **do**
      if $p_j = \emptyset$ **then**
        $\boldsymbol{x}^{(t)}[(j,a)] \coloneqq b_j^{(t)}[a]$;
      else
        $\boldsymbol{x}^{(t)}[(j,a)] \coloneqq \boldsymbol{x}^{(t)}[p_j] \cdot b_j^{(t)}[a]$;
  return $\boldsymbol{x}^{(t)} \in \mathbb{R}^{\Sigma}$;

ObserveUtility($\boldsymbol{u}^{(t)} \in \mathbb{R}^{\Sigma}$):
  $V^{(t)}[\perp] \coloneqq 0$;
  for *each node in the tree* $p \in \mathcal{J} \cup \mathcal{K}$ *in bottom-up order* **do**
    if $p \in \mathcal{J}$ **then**
      Let $j = p$;
      $V^{(t)}[j] \coloneqq \sum_{a \in \mathcal{A}_j} b_j^{(t)}[a] \cdot (\boldsymbol{u}^{(t)}[(j,a)] + V^{(t)}[\rho(j,a)])$;
    else
      Let $k = p$;
      $V^{(t)}[k] \coloneqq \sum_{s \in \mathcal{S}_k} V^{(t)}[\rho(k,s)]$;
  for *each decision point* $j \in \mathcal{J}$ **do**
    for *each action* $a \in \mathcal{A}_j$ **do**
      $\boldsymbol{u}_j^{(t)}[a] \coloneqq \boldsymbol{u}^{(t)}[(j,a)] + V^{(t)}[\rho(j,a)]$;
    $\mathfrak{R}_j$.ObserveUtility($\boldsymbol{u}_j^{(t)}$);

Room for improvement?

- CFR abstracts away the construction of the observed utility
- This is expensive to compute as it requires a full game traversal
- Can we **estimate** the utility?

Room for improvement?

- CFR abstracts away the construction of the observed utility
- This is expensive to compute as it requires a full game traversal
- Can we **estimate** the utility?

- Even if we had the observed utility, we still need to compute counterfactuals and perform an update to each regret minimizer
- Can we only update a small number of infosets by constructing a **sparse** estimator?

Setup: zero-sum games in extensive form

- As usual, we focus on zero-sum games in extensive form
- The goal is to compute approximate minimax (Nash) equilibria
- Most of the techniques extend to regret minimization in general-sum games

$$\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{y}\in\mathcal{Y}} \sum_{z\in\mathcal{Z}} u_2(z)\boldsymbol{x}[\sigma_1(z)]\boldsymbol{y}[\sigma_2(z)]\boldsymbol{c}[\sigma_c(z)],$$

Sequence form

Chance player



Utility computation boils down to a matrix-vector product

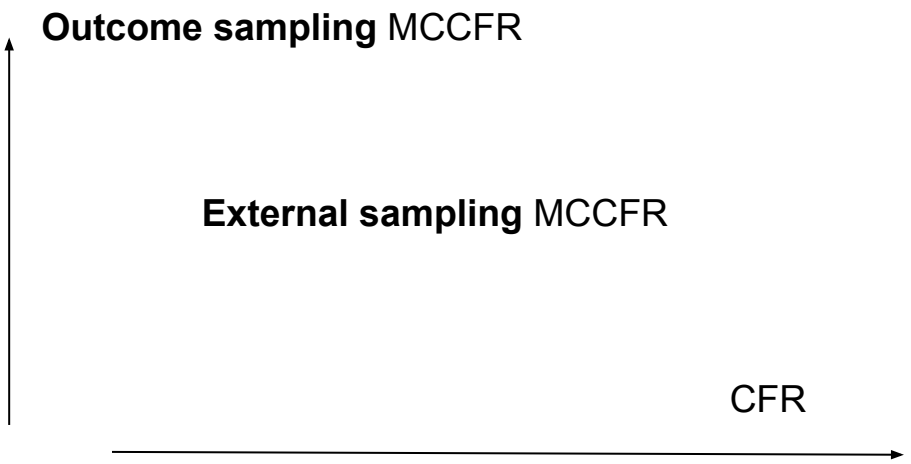
The basic tradeoff in stochastic regret minimization

We want simultaneously

- Sparse and cheap utility estimators
- Without incurring a large degradation in the regret bound

Two main utility estimators:

- External sampling
- Outcome sampling



External sampling

- We sample independently a *pure strategy* of both the opponent and the chance player
- The resulting utility estimator is **unbiased**
- It's called external sampling because it only samples the opponent and the chance player

$$\boldsymbol{x} \mapsto u_1(\boldsymbol{x}, \tilde{\boldsymbol{y}}^{(t)}, \tilde{\boldsymbol{c}}),$$



Computing the utility estimator is much cheaper than computing the exact utility vector

Sublinear algorithms: the case of normal-form games

- What's the complexity of running one iteration of self-play in normal-form games?
- It is already prohibitive when the payoff matrix is dense

$$O\left(\frac{\mathrm{nnz}(A)}{\epsilon}\right)$$

- External sampling reduces the per-iteration cost to *linear*
- This is essentially the famous **sublinear** algorithm of Grigoriadis and Khachiyan (1995)
- Finding an approximate equilibrium can be done without even reading the input!

$$O\left(\frac{m_1 + m_2}{\epsilon^2}\right)$$

External sampling in extensive-form games

An external sampling estimator can be constructed as follows.

- We start from the root and make our way toward to terminal nodes
- If the node belongs to either the opponent or the chance player, we sample an action based on their strategies
- Otherwise we branch over all actions
- We only need to update information sets that the opponent/chance plays to reach in the sampled strategy—typically a small fraction

Outcome sampling

- Outcome sampling further reduces the per-iteration cost by sampling every player's strategy, not just the opponent and chance
- It picks a terminal node with probability

$$\mathbb{P}[\tilde{z}^{(t)} = z] = \boldsymbol{r}^{(t)}[\sigma_1(z)]\boldsymbol{y}^{(t)}[\sigma_2(z)]\boldsymbol{c}[\sigma_c(z)] \quad \forall z \in \mathcal{Z}.$$

Reference strategy

- It then produces the utility estimator

$$\tilde{\boldsymbol{u}}_{\mathcal{X}}^{(t)} = \frac{u_1(\tilde{z}^{(t)})}{\boldsymbol{r}^{(t)}[\sigma_1(\tilde{z}^{(t)})]}\boldsymbol{e}_{\tilde{z}^{(t)}},$$

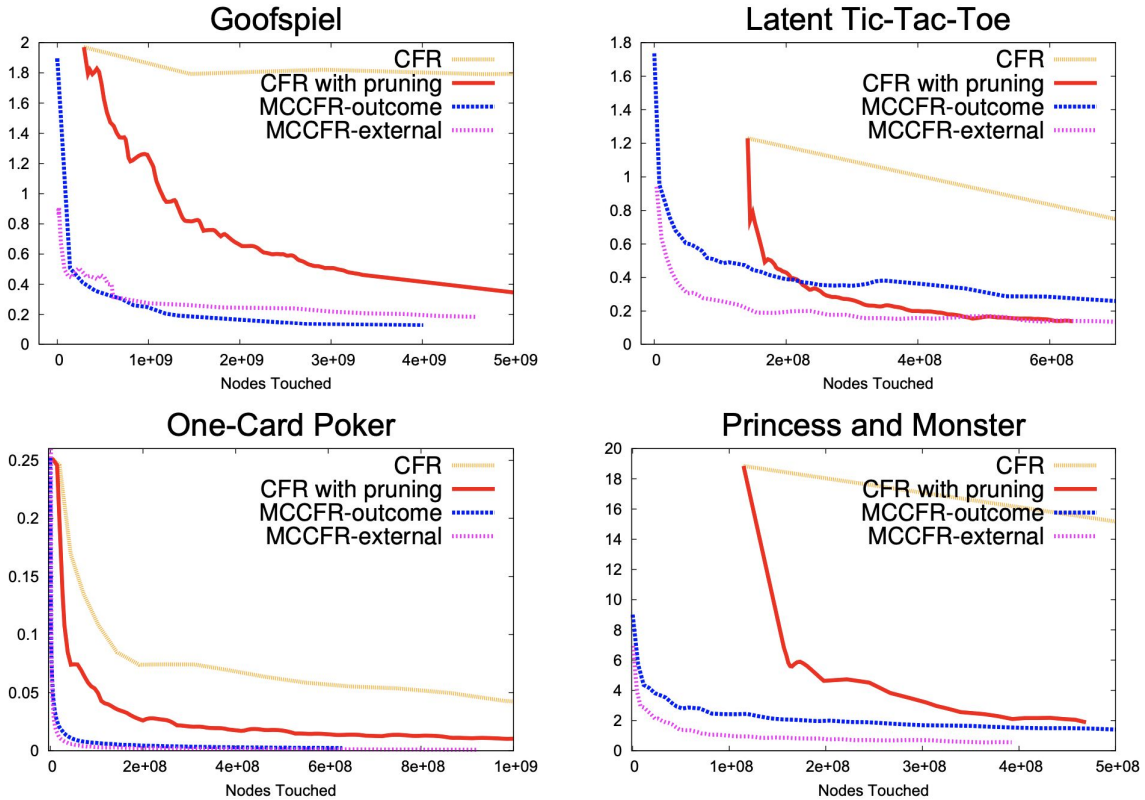
Importance sampling

Outcome sampling

- Outcome sampling is typically significantly cheaper than external sampling
- Sampling a terminal node requires time proportional to the depth of the game tree, which is typically exponentially smaller than the number of nodes
- It then only has to update information sets that are on the path from the root to the sampled terminal node
- The main drawback is the **increased variance**; this can be mitigated by picking a balanced strategy, but only partially so

MCCFR in action

From Lanctot et al. (2009)



Regret analysis of stochastic approximation

- We think of a stochastic regret minimizer that employs a deterministic regret minimizer. The later takes as input an estimated utility
- The main assumption is that the utility estimator is unbiased; more formally,

$$\mathbb{E}[\tilde{\boldsymbol{u}}^{(t)} \mid \tilde{\boldsymbol{u}}^{(1)}, \ldots, \tilde{\boldsymbol{u}}^{(t-1)}] = \boldsymbol{u}^{(t)}.$$



The analysis is very general; it applies beyond MCCFR and external/outcome sampling

Regret analysis of stochastic approximation

Theorem 1.1 (Farina et al., 2020). *Suppose that there are constants M and $\tilde{M}$ such that*

$$|\langle \boldsymbol{x} - \boldsymbol{x}', \boldsymbol{u}^{(t)} \rangle| \le M \text{ and } |\langle \boldsymbol{x} - \boldsymbol{x}', \tilde{\boldsymbol{u}}^{(t)} \rangle| \le \tilde{M} \quad \forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X} \text{ and } t \in [T].$$

Then, for any $\delta \in (0, 1)$, with probability at least $1 - \delta$,

$$\mathrm{Reg}^{(T)} \le \widetilde{\mathrm{Reg}}^{(T)} + (M + \tilde{M}) \sqrt{2T \log \frac{1}{\delta}}.$$



We get sublinear regret with high probability!

Regret analysis of stochastic approximation

Define the martingale difference sequence $d^{(t)} := \langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \boldsymbol{u}^{(t)} \rangle - \langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \tilde{\boldsymbol{u}}^{(t)} \rangle$

$$|d^{(t)}| = |\langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \boldsymbol{u}^{(t)} \rangle - \langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \tilde{\boldsymbol{u}}^{(t)} \rangle| \leq |\langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \boldsymbol{u}^{(t)} \rangle| + |\langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \tilde{\boldsymbol{u}}^{(t)} \rangle| \leq M + \tilde{M}.$$

$$\mathrm{Reg}^{(T)}(\boldsymbol{x}) = \widetilde{\mathrm{Reg}}^{(T)}(\boldsymbol{x}) + \sum_{t=1}^{T} d^{(t)},$$

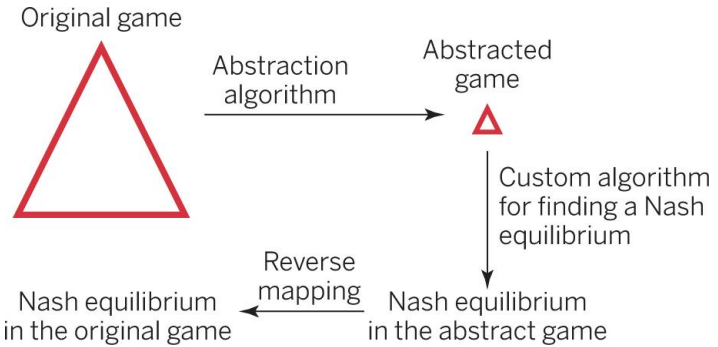
$$\mathbb{P}[\mathrm{Reg}^{(T)}(\boldsymbol{x}) \leq \widetilde{\mathrm{Reg}}^{(T)}(\boldsymbol{x}) + \epsilon] = 1 - \mathbb{P}[\sum_{t=1}^{T} d^{(t)} > \epsilon] \geq 1 - \exp\left(-\frac{2\epsilon^2}{4T(M + \tilde{M})^2}\right).$$

Application to sparse equilibria in normal-form games

- A strategy is called k-*sparse* if it has at most k positive entries
- A well-known result in algorithmic game theory is that there is an equilibrium strategy with *logarithmic sparsity* (Lipton, Markakis, and Mehta, 2003)
- Stochastic regret minimization provides a constructive proof: run MWU by sampling pure strategies; the average is a sparse equilibrium.
- This connection also provides a lower bound for stochastic regret minimization algorithms *even in self-play*; there is no analog of optimistic-type accelerations in the stochastic setting

Automated game abstraction and deep CFR

- Abstraction techniques have been very successful in poker
- The main caveat is that abstraction often requires domain-specific knowledge and may miss strategic nuances of the game
- There is also a chicken-and-egg problem
- **Deep CFR** is a general-purpose abstraction technique that relies on deep neural networks to perform function approximation

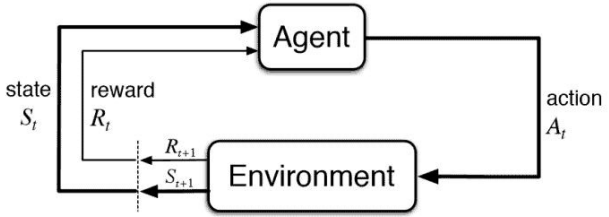


Neural fictitious self-play



- Even though fictitious play was not the most attractive algorithm in the tabular setting, it has been repopularized in large-scale game solving
- Its main advantage is that, unlike more sophisticated methods, it requires only a best response oracle
- Fictitious play is more amenable to deep reinforcement learning techniques
- This motivated Heinrich and Silver (2016) to introduce **neural fictitious self-play**

Neural fictitious self-play

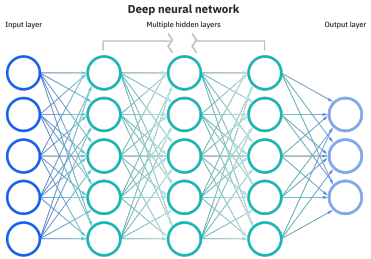


NFSP consists of two basic parts.

- It trains a **value network** to approximate action values in each state
- It maintains a dynamic dataset for training a deep RL algorithm—namely, Q-learning
- This value network is used to determine a best response for the player, as required in fictitious play

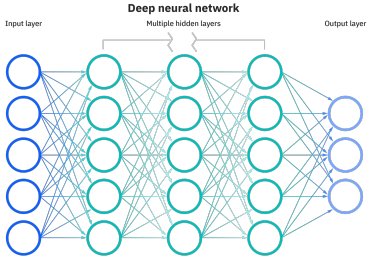
- It also maintains a **policy network** that intends to capture the player's average policy
- This is a supervised learning problem
- This would not be necessary if one had a last-iterate guarantee

Deep CFR



- Deep CFR is a *sound* approximation to CFR (Brown et al., 2019)
- It maintains, for each player, a **regret network** that outputs for each information set an approximation to the counterfactual regrets
- The number of infosets is huge; the hope is that the regret network generalizes well across similar infosets
- Similarly to NFSP, it also uses a **policy network** to approximate the average policy; again, it is only the average of CFR that converges

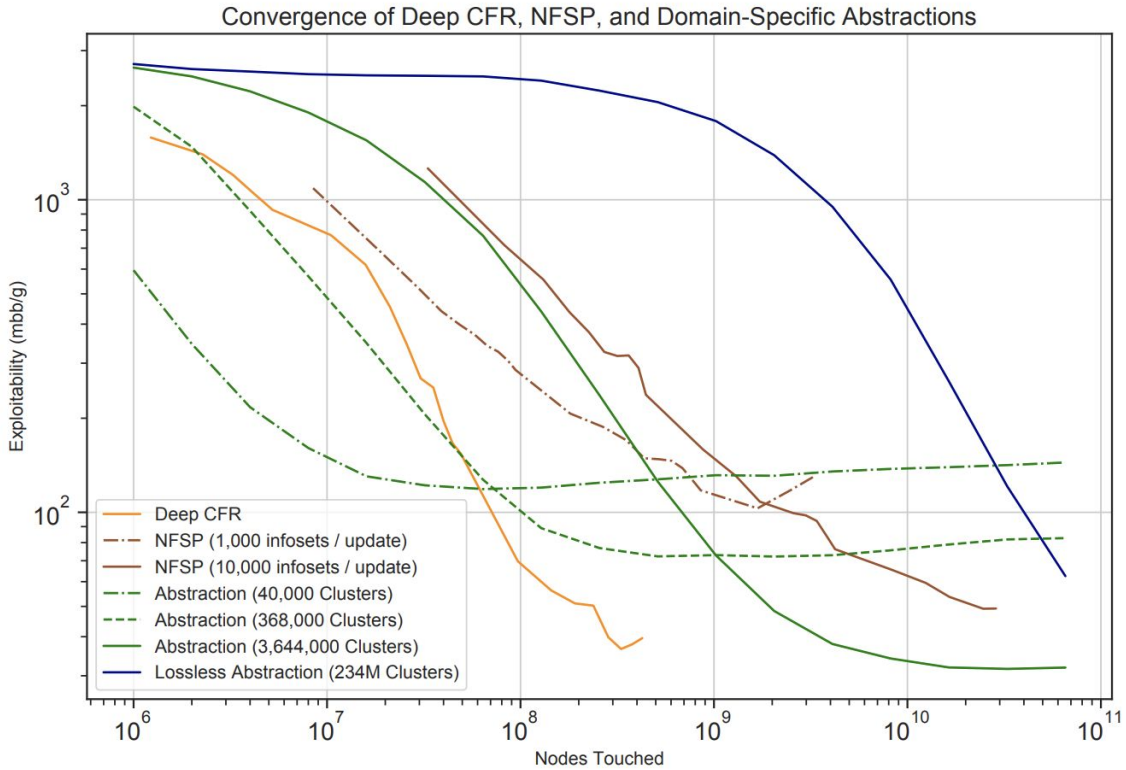
Deep CFR



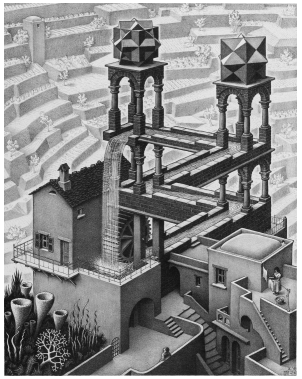
- In each iteration it does multiple partial traversals of the game tree using external sampling MCCFR
- It uses the regret networks together with regret matching to determine each player’s next strategy at a given information set
- It then computes sampled counterfactual regrets from the sampled counterfactual utilities, which are collected in a dataset
- The regret network is then trained using that data
- Game traversals are used to collect data to train the regret network

Deep CFR in action

From Brown et al. (2019)



ESCHER



- The main caveat of outcome sampling MCCFR is that the estimator has large variance due to *importance sampling*
- This is especially problematic when those estimates are produced by a deep neural network, which struggle with outputs of different magnitudes
- ESCHER (McAleer et al., 2023) provides a remedy to this issue by eliminating importance sampling while still guaranteeing theoretical soundness

Reminder on outcome sampling

Outcome sampling picks a terminal node with probability

$$\mathbb{P}[\tilde{z}^{(t)} = z] = \boldsymbol{r}^{(t)}[\sigma_1(z)]\boldsymbol{y}^{(t)}[\sigma_2(z)]\boldsymbol{c}[\sigma_c(z)] \quad \forall z \in \mathcal{Z}.$$

Reference strategy

- It then produces the utility estimator

$$\tilde{\boldsymbol{u}}_{\mathcal{X}}^{(t)} = \frac{u_1(\tilde{z}^{(t)})}{\boldsymbol{r}^{(t)}[\sigma_1(\tilde{z}^{(t)})]}\boldsymbol{e}_{\tilde{z}^{(t)}},$$

Importance sampling

ESCHER

- It has a **fixed** reference strategy for each player; unlike typical outcome sampling MCCFR, it is important for the reference strategy to not change
- It proceeds by sampling a terminal node by following the usual random walk of outcome sampling MCCFR using the reference strategies
- The key difference is that ESCHER estimates the **conditional values**, not the counterfactual values,

$$u_i^{(t)}(a \mid j) = \boldsymbol{x}_{-i}^{(t)}(j) \cdot Q_i^{(t)}(a \mid j),$$

Scaling factor

$$\mathbb{E}[\tilde{u}_i^{(t)}(a \mid j)] = \boldsymbol{r}_i(j) \cdot \boldsymbol{x}_{-i}^{(t)}(j) \cdot Q_i^{(t)}(a \mid j) = \boldsymbol{r}_i(j) \cdot u_i^{(t)}(a \mid j).$$

ESCHER in action

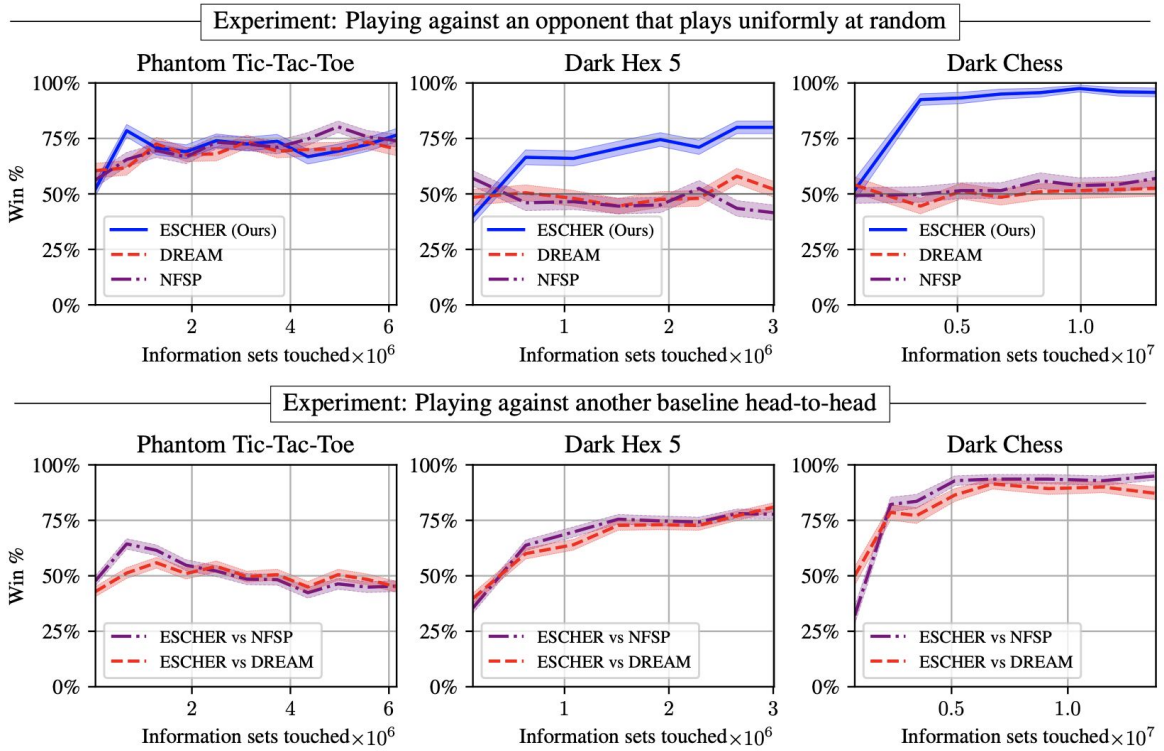
From McAleer et al. (2023)

Game	ESCHER (Ours)	Ablation 1	Ablation 2	DREAM
Phantom Tic-Tac-Toe	$(2.6 \pm 0.1) \times 10^{-1}$	$(4.1 \pm 0.7) \times 10^{1}$	$(1.4 \pm 0.4) \times 10^{7}$	$(4.6 \pm 1.0) \times 10^{7}$
Dark Hex 4	$(1.8 \pm 0.1) \times 10^{-1}$	$(1.3 \pm 0.9) \times 10^{2}$	$(3.1 \pm 1.7) \times 10^{8}$	$(2.8 \pm 2.0) \times 10^{8}$
Dark Hex 5	$(1.3 \pm 0.1) \times 10^{-1}$	$(3.3 \pm 1.6) \times 10^{2}$	$(2.0 \pm 0.6) \times 10^{5}$	$(5.3 \pm 3.9) \times 10^{8}$

Game	ESCHER (Ours)	Ablation 2	DREAM	OS-MCCFR
Leduc	$(5.3 \pm 0.0) \times 10^{0}$	$(3.3 \pm 0.7) \times 10^{2}$	$(2.8 \pm 0.0) \times 10^{2}$	$(2.2 \pm 0.0) \times 10^{3}$
Battleship	$(1.4 \pm 0.0) \times 10^{0}$	$(7.1 \pm 0.3) \times 10^{2}$	$(1.2 \pm 0.0) \times 10^{3}$	$(2.4 \pm 0.0) \times 10^{3}$
Liar's Dice	$(9.0 \pm 0.1) \times 10^{-1}$	$(7.8 \pm 0.9) \times 10^{1}$	$(4.0 \pm 0.8) \times 10^{2}$	$(1.2 \pm 0.1) \times 10^{3}$

ESCHER in action

From McAleer et al. (2023)



ESCHER in action

From McAleer et al. (2023)

